First-order definable counting-only queries*

Jelle Hellings[1], Marc Gyssens[1], Dirk Van Gucht[2], and Yuqing Wu[3]

[1] Hasselt University, Martelarenlaan 42, 3500, Hasselt, Belgium
[2] Indiana University, 150 S. Woodlawn Ave, Bloomington, IN 47405, USA
[3] Pomona College, 185 E 6th St., Claremont, CA 91711, USA

Abstract. For several practical queries on bags of sets of objects, the answer does not depend on the precise composition of these sets, but only on the *number* of sets to which each object belongs. This is the case k=1 for the more general situation where the query answer only depends on the number of sets to which each *group* of at most k objects belongs. We call such queries k-counting-only. Here, we focus on k-SyCALC, k-counting-only queries that are first-order definable. As k-SyCALC is semantically defined, however, it is not surprising that it is already undecidable whether a first-order query is in 1-SyCALC. Therefore, we introduce SimpleCALC-k, a syntactically defined (strict) fragment of k-SyCALC. It turns out that many practical queries in k-SyCALC can already be expressed in SimpleCALC-k. We prove that the k-counting-only queries form a non-collapsing hierarchy: for every k, there exist (k+1)-counting-only queries that are not k-counting-only. This result specializes to both SimpleCALC-k and k-SyCALC. Finally, we establish a strong dichotomy between 1-SyCALC and SimpleCALC-k on the one hand and 2-SyCALC on the other hand by showing that satisfiability, validity, query containment, and query equivalence are decidable for the former two languages, but not for the latter one.

1 Introduction

Often, (parts of) queries can be viewed as operating on a bag of sets, or, equivalently, on transaction databases [8], bipartite graphs, or binary many-to-many relations. As an example, consider the bag-of-sets dataset of Figure 1, *left*, in which each set represents a course and contains the students taking that course. This bag of sets can alternatively be interpreted as the *bipartite graph*, shown in Figure 1, *right*. Many practical queries on bags of sets turn out to be *counting-only*: in order to answer them, it is not necessary to know to which sets each object belongs, but only to *how many* sets each object belongs. As examples, consider the queries 'return students who take at least 2 courses', expressed by

$$\mathtt{Q}_1 = \{\langle x \rangle \mid \mathrm{count}(x) \geq 2\},$$

* This material is based on work supported by the National Science Foundation under Grant No. NSF 1438990.

Fig. 1. *Left*, a bag-of-sets dataset. *Right*, same dataset represented as bipartite graph.

and 'return pairs of students who take the same number of courses', expressed by

$$\mathtt{Q}_2 = \{\langle x, y\rangle \mid (x \neq y) \wedge \mathrm{count}(x) = \mathrm{count}(y)\}.$$

In the above expressions, "count$(\cdot)$" counts the number of sets (here, courses) to which the argument (here, a student) belongs. Clearly, one need not know which courses each student takes to answer $\mathtt{Q}_1$ or $\mathtt{Q}_2$, but only *how many* courses each student takes. Next, consider the queries 'return pairs of distinct students which take a common course', expressed by

$$\mathtt{Q}_3 = \{\langle x, y\rangle \mid (x \neq y) \wedge \mathrm{count}(x, y) \geq 1\},$$

and 'return pairs of distinct students which take the same courses', expressed by

$$\mathtt{Q}_4 = \{\langle x, y\rangle \mid (x \neq y) \wedge \mathrm{count}(x, y) = \mathrm{count}(x) \wedge \mathrm{count}(x, y) = \mathrm{count}(y)\}.$$

Notice that $\mathtt{Q}_3$ is a basic intersection query and $\mathtt{Q}_4$ is a basic equivalence query. Both can be answered by counting not only *(i)* how many courses each student takes, but also *(ii)* how many courses each *pair* of students share. For $k \geq 0$, we call a query *k-counting-only* if it can be answered by only counting to how many sets each *group* of at most k objects belongs. Hence, $\mathtt{Q}_1$ and $\mathtt{Q}_2$ are 1-counting-only, while $\mathtt{Q}_3$ and $\mathtt{Q}_4$ are 2-counting-only. Similarly, the Boolean query 'does there exist a course taken by 3 students', expressed by

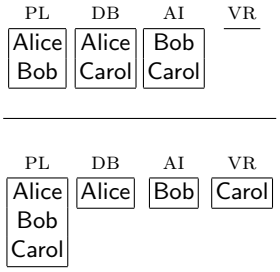
$$\mathtt{Q}_5 = \{\langle\rangle \mid \exists x \exists y \exists z\ ((x \neq y \wedge x \neq z \wedge y \neq z) \wedge \mathrm{count}(x, y, z) \geq 1)\},$$

is 3-counting-only. In contrast, the Boolean query 'there are at least 3 courses', expressed by

$$\mathtt{Q}_6 = \{\langle\rangle \mid \mathrm{count}() \geq 3\}.$$

can already be answered at the scheme level, and is therefore 0-counting-only.

Observe that the counting-only queries $\mathtt{Q}_3$ and $\mathtt{Q}_4$ only differ in the use of the *generalized quantifiers* 'takes some' versus 'takes all and only'. Similar familiar families of counting-only queries can be formulated using other generalized quantifiers such as 'takes only', 'takes all', 'takes no', 'takes at least k', and 'takes all but k'. Such queries are not only of relevance in the study of generalized quantifiers [4,19], but also play an obvious central role in the *frequent itemset problem* [8]. In essence, bag-of-set-like data models and counting-only queries can also be found in the *differential constraints* of Sayrafi et al. [17], *citation analysis and bibliometrics* [5], the *symmetric Boolean functions* of Quine [16,11], *finite*



$$\begin{aligned}
\text{count}() &= 4;\\
\text{count}(\mathsf{A}) &= 2;\\
\text{count}(\mathsf{B}) &= 2;\\
\text{count}(\mathsf{C}) &= 2;\\
\text{count}(\mathsf{A},\mathsf{B}) &= 1;\\
\text{count}(\mathsf{A},\mathsf{C}) &= 1;\\
\text{count}(\mathsf{B},\mathsf{C}) &= 1.
\end{aligned}$$

Fig. 2. *Left*, Bags of sets $\mathbf{S}_1$ (top) and $\mathbf{S}_2$ (bottom), both assigning students to four courses. *Right*, Count-information shared between $\mathbf{S}_1$ and $\mathbf{S}_2$.

set combinatorics [2], and the *data spaces* of Fletcher et al. [7], either explicitly or implicitly.

A more formal way to capture the notion of k-counting-only query is that such queries cannot distinguish between bags of sets which share the same up-to-k counting information. Consider, e.g., the bags of sets $\mathbf{S}_1$ and $\mathbf{S}_2$ of Figure 2. Clearly, $\mathbf{S}_1$ and $\mathbf{S}_2$ agree on all up-to-2 counting information, but disagree on count(Alice, Bob, Carol). Hence, $\mathtt{Q}_1$–$\mathtt{Q}_4$ and $\mathtt{Q}_6$ yield the same result on $\mathbf{S}_1$ and $\mathbf{S}_2$, whereas $\mathtt{Q}_5$, which is 3-counting-only, evaluates to `false` on $\mathbf{S}_1$ and `true` on $\mathbf{S}_2$.

Finally, notice that the concept of counting-only query applies to more general data models than the bag-of-sets model. Consider, e.g., a database with a student-course relation SC and a department-course relation DC, with the obvious meaning. On this database, query

$$\mathtt{P} = \{\langle x, y\rangle \mid \text{count}(\{z \mid SC(x,z) \wedge DC(y,z)\}) = \text{count}(\{z \mid SC(x,z)\})\}$$

returns student-department pairs in which the student only takes courses offered by that department. This query, conceptually similar to $\mathtt{Q}_4$ above, certainly has a counting-only flavor.

Motivated by the above, we believe that the class of counting-only queries deserves a broader understanding. Our notion of k-counting-only queries, $k \geq 0$, significantly generalizes the notion of counting-only queries of Gyssens et al. [11], which only corresponds with our case $k = 1$.

As many interesting counting-only queries are first-order definable, including $\mathtt{Q}_1$ and $\mathtt{Q}_3$–$\mathtt{Q}_6$, we study more specifically the class of first-order definable counting-only queries on the bags-of-sets data model. To do so, we use (a variation of) the two-sorted first-order logic SyCALC of Gyssens et al. In this logic, we have object variables, set name variables, and a set-membership relation relating objects and set names. Our main results are as follows:

1. We semantically define the class of k-counting-only queries, and show that they include many practically relevant first-order-definable queries.
2. We syntactically define the class SimpleCALC-k, $k \geq 0$, a fragment of the first-order-definable queries. All queries in this class turn out to be k-counting-

only. We show that they capture many practical queries in k-SyCALC, the k-counting-only queries in SyCALC. This is in particular the case for those that can be written using simple "count($\cdot$)" terms, such as $\mathtt{Q}_1$ and $\mathtt{Q}_3$–$\mathtt{Q}_6$.

3. We establish that the k-counting-only queries form a non-collapsing hierarchy: for every k, $k \geq 0$, there are $(k{+}1)$-counting-only queries that are not k-counting-only. This result specializes to k-SyCALC and SimpleCALC-k.

4. We show that 1-SyCALC and SimpleCALC-k, $k \geq 0$, have the finite model property and use that to prove that satisfiability (and, hence, validity, query containment, and query equivalence) is decidable for these classes. We also establish that satisfiability is NEXPTIME-hard for SimpleCALC-k. In contrast, satisfiability for 2-SyCALC is shown to be undecidable. Hence, there is a strong dichotomy between 1-SyCALC and SimpleCALC-k, $k \geq 0$, on the one hand and 2-SyCALC on the other hand. Moreover, the decidability of 1-SyCALC and SimpleCALC-k, $k \geq 0$, sets them apart from many other fragments of first-order logic. In particular, this result identifies a large "well-behaved" fragment of first-order logic in which many practical queries can be expressed, and other than the usual classes of "well-behaved" first-order queries such as the conjunctive queries, the monadic first-order logic, and the two-variable fragments of first-order logic [1,14,3,9,10].

2 Bags of sets and counting-only queries

Let $\mathcal{D}$ and $\mathcal{N}$ be two disjoint infinitely enumerable domains of objects and names. We represent finite bags of finite sets by structures, as follows:

Definition 2.1. *A* structure $\mathbf{S}$ *is a pair* $\mathbf{S} = (\mathbf{N}, \gamma)$*, with* $\mathbf{N} \subset \mathcal{N}$ *a finite set of* set names *and* $\gamma \subset \mathcal{D} \times \mathbf{N}$ *a finite* set-membership *relation. For* $\mathrm{N} \in \mathbf{N}$*,* $\mathrm{objects}(\mathrm{N}; \mathbf{S}) = \{\mathsf{o} \mid (\mathsf{o}, \mathrm{N}) \in \gamma\}$ *is the set of objects that are a* member *of the set named* N*. We write* $\mathrm{adom}(\mathbf{S}) = \bigcup_{\mathrm{N} \in \mathcal{N}} \mathrm{objects}(\mathrm{N}; \mathbf{S})$ *for the* active domain *of* $\mathbf{S}$*. If* $\mathrm{A} \subseteq \mathcal{D}$*, then* $\mathbf{S}|_{\mathrm{A}}$ *denotes the structure* $(\mathbf{N}, \gamma \cap (\mathrm{A} \times \mathbf{N}))$*.*

Structures explicitly define the set $\mathbf{N}$ of set names they use, whereas objects are only defined via the set-membership function γ. In this way, $\mathbf{N}$ allows the representation of empty sets:

Example 2.2. The bag-of-sets dataset of Figure 1 is represented by the structure $\mathbf{S}_1 = (\mathbf{N}, \gamma)$ with $\mathbf{N} = \{\mathrm{PL}, \mathrm{DB}, \mathrm{AI}\}$ and $\gamma = \{(\mathsf{Alice}, \mathrm{PL}), (\mathsf{Bob}, \mathrm{PL}), (\mathsf{Alice}, \mathrm{DB}), (\mathsf{Bob}, \mathrm{DB}), (\mathsf{Carol}, \mathrm{DB}), (\mathsf{Dan}, \mathrm{AI})\}$. If we were to add course VR to $\mathbf{N}$ without changing γ, this would mean that VR is offered but no student takes it.

A *query* q maps a structure to a relation of fixed arity over objects. We write $[\![q]\!]_{\mathbf{S}}$ to denote the *evaluation* of q on structure $\mathbf{S}$. If the arity of q is 0, then q is *Boolean*. The only two relations of arity 0, $\emptyset$ and $\{\langle\rangle\}$, represent `false` and `true`, respectively. In the Introduction, we showed that many queries on bags of sets cannot distinguish structures with the same up-to-k count information, for some k, $k \geq 0$. We formalize this next:

Definition 2.3. *Let* $\mathbf{S} = (\mathbf{N}, \gamma)$ *be a structure and* $\mathrm{I} \subset \mathcal{D}$ *a finite set of objects, often referred to as an* itemset. *The* cover *of* I *in* $\mathbf{S}$ *is defined by* $\mathrm{cover}(\mathrm{I}; \mathbf{S}) = \{\mathrm{N} \mid (\mathrm{N} \in \mathbf{N}) \wedge (\mathrm{I} \subseteq \mathrm{objects}(\mathrm{N}; \mathbf{S}))\}$. *The* support *of* I *in* $\mathbf{S}$ *is defined by* $[\![\mathrm{count}(\mathrm{I})]\!]_{\mathbf{S}} = |\mathrm{cover}(\mathrm{I}; \mathbf{S})|$. *Structures* $\mathbf{S}_1$ *and* $\mathbf{S}_2$ *are* exactly-k-counting-equivalent *if* $[\![\mathrm{count}(\mathrm{I})]\!]_{\mathbf{S}_1} = [\![\mathrm{count}(\mathrm{I})]\!]_{\mathbf{S}_2}$ *for every itemset* I *with* $|\mathrm{I}| = k$. *Structures* $\mathbf{S}_1$ *and* $\mathbf{S}_2$ *are* k-counting-equivalent *if they are exactly-j-counting-equivalent for all j,* $0 \leq j \leq k$.[4]

Structures are exactly-0-counting-equivalent if they have the same number of set names. Hence, for all k, $k \geq 0$, k-counting-equivalent structures have the same number of set names.

Example 2.4. Consider the structures $\mathbf{S}_1$ and $\mathbf{S}_2$ in Figure 2. Both have four set names representing courses. In both $\mathbf{S}_1$ and $\mathbf{S}_2$, each student takes two courses, and each pair of distinct students shares one common course. Since the itemset $\{\mathsf{Alice}, \mathsf{Bob}, \mathsf{Carol}\}$ has no cover in $\mathbf{S}_1$, but is covered by PL in $\mathbf{S}_2$, we conclude that $\mathbf{S}_1$ and $\mathbf{S}_2$ are 2-counting-equivalent, but not 3-counting-equivalent.

We are now ready to define k-counting-only queries:

Definition 2.5. *A query q is* k-counting-only *if, for every pair of k-counting-equivalent structures* $\mathbf{S}_1$ *and* $\mathbf{S}_2$, *we have* $[\![q]\!]_{\mathbf{S}_1} = [\![q]\!]_{\mathbf{S}_2}$. *A query is* counting-only *if there exists k, $k \geq 0$, such that the query is k-counting-only.*[5]

Example 2.6. As mentioned in the Introduction, $\mathtt{Q}_1$ and $\mathtt{Q}_2$ are 1-counting-only, $\mathtt{Q}_3$ and $\mathtt{Q}_4$ are 2-counting-only, $\mathtt{Q}_5$ is 3-counting-only, and $\mathtt{Q}_6$ is 0-counting-only. Query $\mathtt{Q}_5$ is not 2-counting-only, since, on the 2-counting-equivalent structures $\mathbf{S}_1$ and $\mathbf{S}_2$ in Figure 2, it returns different results. Notice that $\mathtt{Q}_2$ involves pairs of objects despite being 1-counting-only. To illustrate that this generalizes, consider

$$\begin{aligned}\mathtt{Q}_7 = \{\langle\rangle \mid \exists x \exists y_1 \exists y_2\ & (x \neq y_1) \wedge (x \neq y_2) \wedge (y_1 \neq y_2) \wedge \\ & \mathrm{count}(y_1) = \mathrm{count}(x, y_1) \wedge \mathrm{count}(y_2) = \mathrm{count}(x, y_2) \wedge \\ & \mathrm{count}(x) = \mathrm{count}(x, y_1) + \mathrm{count}(x, y_2) - \mathrm{count}(x, y_1, y_2)\}.\end{aligned}$$

On the student-courses examples, $\mathtt{Q}_7$ returns `true` if there is a student who takes exactly the courses taken by a pair of distinct other students combined. Clearly, it is 3-counting-only. However, $\mathtt{Q}_7$ is also 2-counting-only, as it is equivalent to

$$\begin{aligned}\mathtt{Q}'_7 = \{\langle\rangle \mid \exists x \exists y_1 \exists y_2\ & (x \neq y_1) \wedge (x \neq y_2) \wedge (y_1 \neq y_2) \wedge \\ & \mathrm{count}(y_1) = \mathrm{count}(x, y_1) \wedge \mathrm{count}(y_2) = \mathrm{count}(x, y_2) \wedge \\ & \mathrm{count}(x) = \mathrm{count}(x, y_1) + \mathrm{count}(x, y_2) - \mathrm{count}(y_1, y_2)\}.\end{aligned}$$

So, some 2-counting-only queries can be used to reason on more than two objects.

[4] Gyssens et al. [11] use the tem *incidence* to refer to the support of a single object, and *incidence-equivalence* to refer to 1-counting-equivalence.

[5] Gyssens et al. [11] use the term *counting-only* to denote the first-order definable queries that are 1-counting-only.

We now show that k-counting information can be used to express the existence of any set-membership relation between at most k objects. To do so, we use the notion of *generalized support*, borrowed from Calders et al. [6].

Definition 2.7. *The* generalized cover *of itemsets* I *and* E *in structure* $\mathbf{S} = (\mathbf{N}, \gamma)$ *is defined by* $\text{gcover}(\text{I}; \text{E}; \mathbf{S}) = \{\text{N} \mid (\text{N} \in \mathbf{N}) \wedge (\text{I} \subseteq \text{objects}(\text{N}; \mathbf{S})) \wedge (\text{objects}(\text{N}; \mathbf{S}) \cap \text{E} = \emptyset)\}$ *and the* generalized support *of* I *and* E *in* $\mathbf{S}$ *is defined by* $[\![\text{gcount}(\text{I}; \text{E})]\!]_{\mathbf{S}} = |\text{gcover}(\text{I}; \text{E}; \mathbf{S})|$.

Observe that $\text{I} \cap \text{E} \neq \emptyset$ implies that $\text{gcover}(\text{I}; \text{E}; \mathbf{S}) = \emptyset$ and $[\![\text{gcount}(\text{I}; \text{E})]\!]_{\mathbf{S}} = 0$. Using the inclusion-exclusion principle [6], we can show that generalized-support terms $[\![\text{gcount}(\text{I}; \text{E})]\!]_{\mathbf{S}}$ are fully expressible using $|\text{I} \cup \text{E}|$-support terms only:

Proposition 2.8. *Let* $\mathbf{S}_1$ *and* $\mathbf{S}_2$ *be* k*-counting-equivalent structures and let* I, E *be itemsets with* $|\text{I} \cup \text{E}| \leq k$*. We have* $[\![\text{gcount}(\text{I}; \text{E})]\!]_{\mathbf{S}_1} = [\![\text{gcount}(\text{I}; \text{E})]\!]_{\mathbf{S}_2}$.

Allowing basic gcount$(\cdot)$ terms[6] of the form $\text{gcount}(\mathcal{X}; \mathcal{Y}) \sim c$, with $\mathcal{X}$ and $\mathcal{Y}$ sets of object variables, "$\sim$" a comparison, and c a constant, often simplifies the expression of counting-only queries.

Example 2.9. Since $\text{count}(\mathcal{X}) = \text{gcount}(\mathcal{X}; \emptyset)$, $\mathtt{Q}_1$, $\mathtt{Q}_3$, $\mathtt{Q}_5$, and $\mathtt{Q}_6$ can be expressed with basic gcount$(\cdot)$ terms. Query $\mathtt{Q}_2$ cannot be rewritten with basic gcount$(\cdot)$ terms, because it is not first-order definable [15] (see also Proposition 5.3). Query $\mathtt{Q}_4$ is equivalent to $\mathtt{Q}'_4 = \{\langle x, y\rangle \mid (x \neq y) \wedge \text{gcount}(x; y) = 0 \wedge \text{gcount}(y; x) = 0\}$. Finally, $\mathtt{Q}_7$ and $\mathtt{Q}'_7$ are equivalent to

$$\mathtt{Q}''_7 = \{\langle\rangle \mid \exists x \exists y_1 \exists y_2\ (x \neq y_1) \wedge (x \neq y_2) \wedge (y_1 \neq y_2) \wedge \text{gcount}(x; y_1, y_2) = 0 \wedge \text{gcount}(y_1; x) = 0 \wedge \text{gcount}(y_2; x) = 0\}.$$

3 A first-order logic for bag-of-sets structures

We now study the relationships between counting-only queries and first-order definable queries. To query bag-of-sets structures, we use a two-sorted variant of first-order logic denoted SyCALC, based on the work of Gyssens et al. [11].[7] *Partial SyCALC formulae* are defined by the grammar

$$e := \Gamma(x, X) \mid x = y \mid X = Y \mid e \vee e \mid \neg e \mid \exists x\ e \mid \exists X\ e,$$

in which the lowercase variables x and y represent objects and the uppercase variables X and Y denote set names. We also allow the usual shorthands.

As to the semantics of a partial SyCALC formula e, let $\mathbf{S} = (\mathbf{N}, \gamma)$ be a structure, $\nu_{\mathcal{D}}$ a mapping from object variables to objects in $\mathcal{D}$, and $\nu_{\mathbf{N}}$ a mapping from

[6] These play a central role in the normal form of 1-counting-only first-order definable queries of Gyssens et al. [11]: $\mathbf{gteq}(\mathtt{o}, c)$ corresponds to $[\![\text{gcount}(\mathtt{o}; \emptyset)]\!]_{\mathbf{S}} \geq c$ and $\mathbf{cogteq}(\mathtt{o}, c)$ to $[\![\text{gcount}(\emptyset; \mathtt{o})]\!]_{\mathbf{S}} \geq |\mathbf{N}| - c$.

[7] Gyssens et al. [11] disallow object comparisons ($x = y$ in the grammar).

set name variables to set names in $\mathbf{N}$. We define the relationship $(\mathbf{S}, \nu_{\mathcal{D}}, \nu_{\mathbf{N}}) \vDash e$, with all free variables of e in the union of the domains of $\nu_{\mathcal{D}}$ and $\nu_{\mathbf{N}}$, as follows:

$$
\begin{array}{ll}
(\mathbf{S}, \nu_{\mathcal{D}}, \nu_{\mathbf{N}}) \vDash \Gamma(x, X) & \text{if } (\nu_{\mathcal{D}}(x), \nu_{\mathbf{N}}(X)) \in \gamma; \\
(\mathbf{S}, \nu_{\mathcal{D}}, \nu_{\mathbf{N}}) \vDash x = y & \text{if } \nu_{\mathcal{D}}(x) = \nu_{\mathcal{D}}(y); \\
(\mathbf{S}, \nu_{\mathcal{D}}, \nu_{\mathbf{N}}) \vDash X = Y & \text{if } \nu_{\mathbf{N}}(X) = \nu_{\mathbf{N}}(Y); \\
(\mathbf{S}, \nu_{\mathcal{D}}, \nu_{\mathbf{N}}) \vDash e_1 \vee e_2 & \text{if } (\mathbf{S}, \nu_{\mathcal{D}}, \nu_{\mathbf{N}}) \vDash e_1 \text{ or } (\mathbf{S}, \nu_{\mathcal{D}}, \nu_{\mathbf{N}}) \vDash e_2; \\
(\mathbf{S}, \nu_{\mathcal{D}}, \nu_{\mathbf{N}}) \vDash \neg e & \text{if } (\mathbf{S}, \nu_{\mathcal{D}}, \nu_{\mathbf{N}}) \nvDash e; \\
(\mathbf{S}, \nu_{\mathcal{D}}, \nu_{\mathbf{N}}) \vDash \exists x\ e & \text{if there exists } \mathsf{o} \in \mathcal{D} \text{ with } (\mathbf{S}, \nu_{\mathcal{D}}[x \mapsto \mathsf{o}], \nu_{\mathbf{N}}) \vDash e; \\
(\mathbf{S}, \nu_{\mathcal{D}}, \nu_{\mathbf{N}}) \vDash \exists X\ e & \text{if there exists } \mathrm{N} \in \mathbf{N} \text{ with } (\mathbf{S}, \nu_{\mathcal{D}}, \nu_{\mathbf{N}}[X \mapsto \mathrm{N}]) \vDash e.
\end{array}
$$

Above, $M[\alpha \mapsto \beta]$ denotes M modified by mapping α to β.

Let e be a partial SyCALC formula with free object variables $x_1, \ldots, x_m$ and free set name variables $X_1, \ldots, X_n$, and let $\mathbf{S} = (\mathbf{N}, \gamma)$ be a structure. We define the *evaluation* of e on $\mathbf{S}$ by $[\![e]\!]_{\mathbf{S}} = \{\langle \mathsf{o}_1, \ldots, \mathsf{o}_m, \mathrm{N}_1, \ldots, \mathrm{N}_n \rangle \mid (\mathbf{S}, \nu_{\mathcal{D}}, \nu_{\mathbf{N}}) \vDash e\}$ in which $\nu_{\mathcal{D}} = \{x_1 \mapsto \mathsf{o}_1, \ldots, x_m \mapsto \mathsf{o}_m\}$ and $\nu_{\mathbf{N}} = \{X_1 \mapsto \mathrm{N}_1, \ldots, X_n \mapsto \mathrm{N}_n\}$. A *SyCALC query* is a partial SyCALC formula without free set name variables.[8]

Example 3.1. Queries $\mathtt{Q}_1$ and $\mathtt{Q}_3$–$\mathtt{Q}_7$ are all expressible in SyCALC:

$$
\begin{aligned}
\mathtt{Q}_1 &= \{\langle x \rangle \mid \exists X_1 \exists X_2\ ((X_1 \neq X_2) \wedge \Gamma(x, X_1) \wedge \Gamma(x, X_2))\}; \\
\mathtt{Q}_3 &= \{\langle x, y \rangle \mid (x \neq y) \wedge \exists X\ (\Gamma(x, X) \wedge \Gamma(y, X))\}; \\
\mathtt{Q}_4 &= \{\langle x, y \rangle \mid (x \neq y) \wedge \forall X\ (\Gamma(x, X) \iff \Gamma(y, X))\}; \\
\mathtt{Q}_5 &= \{\langle\rangle \mid \exists X \exists x \exists y \exists z\ ((x \neq y) \wedge (x \neq z) \wedge (y \neq z) \wedge \\
&\qquad\qquad \Gamma(x, X) \wedge \Gamma(y, X) \wedge \Gamma(z, X))\}; \\
\mathtt{Q}_6 &= \{\langle\rangle \mid \exists X_1 \exists X_2 \exists X_3\ ((X_1 \neq X_2) \wedge (X_1 \neq X_3) \wedge (X_2 \neq X_3))\}; \\
\mathtt{Q}_7 &= \{\langle\rangle \mid \exists x \exists y_1 \exists y_2\ ((x \neq y_1) \wedge (x \neq y_2) \wedge (y_1 \neq y_2) \wedge \\
&\qquad\qquad (\forall X\ (\Gamma(x, X) \iff (\Gamma(y_1, X) \vee \Gamma(y_2, X)))))\}.
\end{aligned}
$$

Not all counting-only queries are in SyCALC. An example is the 1-counting-only query $\mathtt{Q}_2$ [15] (see also Proposition 5.3). Also, not all SyCALC queries are counting-only. To show this, we must exhibit a SyCALC query $\mathtt{Q}$ and, for every k, $k \geq 0$, a pair of k-counting-equivalent structures $\mathbf{S}_{1,k}$ and $\mathbf{S}_{2,k}$, such that $\mathtt{Q}$ can distinguish $\mathbf{S}_{1,k}$ and $\mathbf{S}_{2,k}$. To do so, we generalize the ideas underlying Example 2.4:

Proposition 3.2. *Let* A *be a finite nonempty itemset, and* $\mathbf{S}_{1,\mathrm{A}}$ *and* $\mathbf{S}_{2,\mathrm{A}}$ *structures respectively representing the bags of sets* $\{\mathrm{T} \mid \mathrm{T} \subseteq \mathrm{A}$ *and* $\mathrm{even}(|\mathrm{A} - \mathrm{T}|)\}$ *and* $\{\mathrm{T} \mid \mathrm{T} \subseteq \mathrm{A}$ *and* $\mathrm{odd}(|\mathrm{A} - \mathrm{T}|)\}$. *We have the following:*

(i) $\mathbf{S}_{1,\mathrm{A}}$ *is* $(|\mathrm{A}| - 1)$*-counting-equivalent to* $\mathbf{S}_{2,\mathrm{A}}$.
(ii) $\mathbf{S}_{1,\mathrm{A}}$ *is not exactly-*$|\mathrm{A}|$*-counting-equivalent to* $\mathbf{S}_{2,\mathrm{A}}$.

[8] We also write a SyCALC query e as $\{\langle x_1, \ldots, x_m \rangle \mid e\}$ to show the free object variables and their order explicitly.

(iii) Only one of the structures has a set name to which no objects are related.

Proof. Statement (ii) follows from the observation that only the itemset A has |A| objects, and only $\mathbf{S}_1$ has a set name that covers this itemset. Statement (iii) follows from the observation that $\emptyset$ is represented only in $\mathbf{S}_1$—if even(|A|)—or only in $\mathbf{S}_2$—if odd(|A|). We now turn to Statement (i). Let $k = |\mathrm{A}|$ and $\mathrm{I} \subsetneq \mathrm{A}$ an itemset with $|\mathrm{I}| = m$. We must prove that $[\![\mathrm{count}(\mathrm{I})]\!]_{\mathbf{S}_1} = [\![\mathrm{count}(\mathrm{I})]\!]_{\mathbf{S}_2}$. Consider any itemset T with $\mathrm{I} \subseteq \mathrm{T} \subseteq \mathrm{A}$. Let $|\mathrm{T}| = n$. As T contains the objects of I, there remain $n - m$ unconstrained objects in $\mathrm{A} - \mathrm{I}$. Hence, there are exactly $\binom{k-m}{n-m}$ of such sets T. Thus,

$$\begin{aligned}[\![\mathrm{count}(\mathrm{I})]\!]_{\mathbf{S}_1} &= \sum_{\substack{m \le n \le k,\\ \mathrm{even}(k-n)}} \tbinom{k-m}{n-m} = \sum_{\substack{0 \le j \le k-m,\\ \mathrm{even}(k-m-j)}} \tbinom{k-m}{j} = 2^{k-m-1} \\ &= \sum_{\substack{0 \le j \le k-m,\\ \mathrm{odd}(k-m-j)}} \tbinom{k-m}{j} = \sum_{\substack{m \le n \le k,\\ \mathrm{odd}(k-n)}} \tbinom{k-m}{n-m} = [\![\mathrm{count}(\mathrm{I})]\!]_{\mathbf{S}_2},\end{aligned}$$

completing the proof. □

Using Proposition 3.2, we can now prove the following:

Proposition 3.3. *Not all Boolean SyCALC queries are counting-only.*

Proof. For all k, $k \ge 0$, let $\mathrm{A}_k \subset \mathcal{D}$ be a set of objects with $|\mathrm{A}_k| = k + 1$, and let $\mathbf{S}_{1,\mathrm{A}_k}$ and $\mathbf{S}_{2,\mathrm{A}_k}$ be as in Proposition 3.2. We see that the Boolean SyCALC query

$$\mathtt{Q}_8 = \{\langle\rangle \mid \exists X \forall x\ (\exists Y\ (\Gamma(x, Y)) \implies \Gamma(x, X))\}$$

cannot be counting-only, since $[\![\mathtt{Q}_8]\!]_{\mathbf{S}_{1,\mathrm{A}_k}} = \mathtt{true}$ and $[\![\mathtt{Q}_8]\!]_{\mathbf{S}_{2,\mathrm{A}_k}} = \mathtt{false}$. □

Even though not all counting-only queries are in SyCALC and vice versa, there is a strong connection between both: all basic gcount(·) terms are expressible in SyCALC. E.g., $\mathrm{gcount}(\mathcal{X}; \mathcal{Y}) \ge c$ is expressed by

$$\exists Z_1 \ldots \exists Z_c \Big(\textstyle\bigwedge_{1 \le i < j \le c} (Z_i \ne Z_j) \wedge \bigwedge_{x \in \mathcal{X}} \big(\Gamma(x, Z_1) \wedge \cdots \wedge \Gamma(x, Z_c)\big) \wedge \bigwedge_{y \in \mathcal{Y}} \big(\neg\Gamma(y, Z_1) \wedge \cdots \wedge \neg\Gamma(y, Z_c)\big)\Big).$$

4 QuineCALC and SimpleCALC

In Section 3, we studied the counting-only SyCALC queries, a *semantic* fragment of SyCALC. The observation that the SyCALC expression for $\mathrm{gcount}(\mathcal{X}; \mathcal{Y}) \ge c$ above, which can be used to express most queries we have seen up till now, does not use object quantification inspires us to define the following *syntactic* fragments of SyCALC:

Definition 4.1. *QuineCALC*[9] *consist of all SyCALC queries that do not use object quantification. SimpleCALC consists of all queries that are built from QuineCALC queries using disjunction, negation, and object quantification.*

[9] Gyssens et al. [11] introduced the single-object-variable fragment of QuineCALC as a first-order query language that provides a conservative extension of the symmetric Boolean functions of Quine [16], hence the name.

For $k \geq 0$, k-SyCALC denotes the k-counting-only SyCALC queries; QuineCALC-k denotes the QuineCALC queries with at most k free object variables; and SimpleCALC-k denotes the SimpleCALC queries built from QuineCALC-k queries.

By definition, all queries expressible using basic gcount$(\cdot)$ terms only, such as $\mathtt{Q}_1$ and $\mathtt{Q}_3$–$\mathtt{Q}_7$, are in SimpleCALC. We will show next that all SimpleCALC-k queries are k-counting-only. To do so, we need

Definition 4.2. *Let* $\mathbf{S}_1 = (\mathbf{N}_1, \gamma_1)$ *and* $\mathbf{S}_2 = (\mathbf{N}_2, \gamma_2)$ *be structures, and let* I *be an itemset. Set names* $\mathrm{N}_1 \in \mathbf{N}_1$ *and* $\mathrm{N}_2 \in \mathbf{N}_2$ *are* I-equivalent *if* $\mathrm{objects}(\mathrm{N}_1; \mathbf{S}_1) \cap \mathrm{I} = \mathrm{objects}(\mathrm{N}_2; \mathbf{S}_2) \cap \mathrm{I}$. *A bijection* $b : \mathbf{N}_1 \to \mathbf{N}_2$ *is an* I-preserving mapping *if, for all* $\mathrm{N} \in \mathbf{N}_1$, N *and* $b(\mathrm{N})$ *are* I*-equivalent.*

We can now give an alternative characterization of k-counting equivalence:

Lemma 4.3. *Let* $\mathbf{S}_1 = (\mathbf{N}_1, \gamma_1)$ *and* $\mathbf{S}_2 = (\mathbf{N}_2, \gamma_2)$ *be structures. Then,* $\mathbf{S}_1$ *and* $\mathbf{S}_2$ *are* k*-counting-equivalent if and only if, for every itemset* I, $|\mathrm{I}| \leq k$, *there exists an* I*-preserving mapping* $b : \mathbf{N}_1 \to \mathbf{N}_2$.

Using Lemma 4.3, a straightforward structural induction argument on partial QuineCALC formulae—partial SyCALC formula without object quantification—yields the following:

Lemma 4.4. *Let* e *be a partial QuineCALC formula with* k *free object variables. For every pair of* k*-counting-equivalent structures* $\mathbf{S}_1 = (\mathbf{N}_1, \gamma_1)$, $\mathbf{S}_2 = (\mathbf{N}_2, \gamma_2)$, *every mapping* $\nu_{\mathcal{D}}$ *from free object variables in* e *to an itemset* $\mathrm{I} \subset \mathcal{D}$ *with* $|\mathrm{I}| \leq k$, *every mapping* $\nu_{\mathbf{N}_1}$ *from free set name variables in* e *to* $\mathbf{N}_1$, *and every* I*-preserving mapping* b *from* $\mathbf{S}_1$ *to* $\mathbf{S}_2$, $(\mathbf{S}_1, \nu_{\mathcal{D}}, \nu_{\mathbf{N}_1}) \vDash e \iff (\mathbf{S}_2, \nu_{\mathcal{D}}, b \circ \nu_{\mathbf{N}_1}) \vDash e$.

Lemma 4.4 implies that QuineCALC-k queries are k-counting-only. To extend this to SimpleCALC-k, it suffices to show that

Proposition 4.5. *k-SyCALC is closed under disjunction, negation, and object quantification.*

Corollary 4.6. *All QuineCALC-k and SimpleCALC-k queries are in k-SyCALC.*

5 Counting-only hierarchies

We now have four hierarchies of counting-only queries, for $k \geq 0$: k-counting-only queries, k-SyCALC, QuineCALC-k, and SimpleCALC-k. We show that all four hierarchies are non-collapsing:

Theorem 5.1. *Let* $k \geq 0$.

(i) Every k-counting-only query is also $(k+1)$-counting-only.
(ii) There is QuineCALC-$(k+1)$ query which is not k-counting-only.
(iii) There is a Boolean SimpleCALC-$(k+1)$ query which is not k-counting-only.

Proof. Statement (i) follows immediately from the definition. For Statements (ii) and (iii), let $\mathbf{S}_{1,\mathrm{A}}$ and $\mathbf{S}_{2,\mathrm{A}}$ be the structures of Proposition 3.2 with $|\mathrm{A}| = k+1$. These structures are k-counting-equivalent, but not $(k+1)$-counting-equivalent. For Statement (ii), we consider $e = \exists X \left(\bigwedge_{1 \leq i \leq k+1} \Gamma(x_i, X)\right)$, which is a $(k+1)$-counting-only QuineCALC-$(k+1)$ query by Corollary 4.6. Let t be a $(k+1)$-tuple containing each value of A once. Then, $t \in [\![e]\!]_{\mathbf{S}_1}$, but $t \notin [\![e]\!]_{\mathbf{S}_2}$. Hence, e is not k-counting-only. For Statement (iii), we construct from e the Boolean SimpleCALC-$(k+1)$ query $e' = \exists x_1 \ldots x_{k+1} \left(\left(\bigwedge_{1 \leq j < j' \leq k+1} (x_j \neq x_{j'})\right) \wedge e(x_1, \ldots, x_{k+1})\right)$ Then, $[\![e']\!]_{\mathbf{S}_1} = \mathtt{true}$ and $[\![e']\!]_{\mathbf{S}_2} = \mathtt{false}$. Hence, e' is not k-counting-only. □

Statement (iii) can be interpreted as the Boolean version of Statement (ii). Since QuineCALC-k and SimpleCALC-k queries are also k-SyCALC queries as well as k-counting queries, Theorem 5.1 extends to all four hierarchies.

We now proceed by comparing the fragments *mutually*. The 0-counting-only fragments have straightforward relationships:

Proposition 5.2. *The languages* 0-*SyCALC*, *SimpleCALC*-0, *and* *QuineCALC*-0 *all express exactly the same set of queries.*

We have already argued that the 1-counting-only query $\mathtt{Q}_2$ is not first-order definable [15]. Also the 0-counting-only query

$$\mathtt{Q}_9 = \{\langle\rangle \mid \text{count}() \text{ is } \textit{even}\}$$

is not first-order definable. Consequently, we have:

Proposition 5.3. *There is a Boolean* 0*-counting-only query not expressible in* *SyCALC*.

By Proposition 5.3 and Theorem 5.1 (i), $\mathtt{Q}_9$ also witnesses that, for all k, $k \geq 0$, there is a Boolean k-counting-only queries not expressible in k-SyCALC.

Due to QuineCALC queries not allowing object quantification, all Boolean QuineCALC queries are in QuineCALC-0. Hence, no Boolean query that is k-counting-only, $k \geq 1$, but not $(k$-1)-counting-only is expressible in QuineCALC-k. Hence, it only remains to establish a separation between k-SyCALC and SimpleCALC-k. We first deal with the special case $k = 1$.

Proposition 5.4. *There is a Boolean* 1-*SyCALC query not expressible in* *SimpleCALC*-1.

Proof. The Boolean 1-SyCALC query

$$\mathtt{Q}_{10} = \{\langle\rangle \mid \exists x \exists y \ ((x \neq y) \wedge \exists X \exists Y \ (\Gamma(x, X) \wedge \Gamma(y, Y)))\},$$

which queries for structures with an active domain of at least two objects, is 1-counting-only but not expressible in SimpleCALC-1. □

To establish the separation between k-SyCALC and SimpleCALC-k, $k \geq 2$, we exhibit a 2-SyCALC query, which is not 1-counting-only, that is not expressible in SimpleCALC. Thereto, let

$$\texttt{set-ids} = \forall X \exists x \ (\Gamma(x, X) \wedge \neg \exists Y \ ((X \neq Y) \wedge \Gamma(x, Y)))$$

be the Boolean query specifying that each set in a bag of sets has a distinct identifying object. We first prove that `set-ids` is in 2-SyCALC, but not in 1-SyCALC, despite it using only a single object variable.

Proposition 5.5. *Query* `set-ids` *is* 2*-counting-only, but not* 1*-counting-only.*

Proof. Let $\mathsf{o}_1, \mathsf{o}_2 \in \mathcal{D}$ and $\mathrm{N}_1, \mathrm{N}_2 \in \mathcal{N}$. Let $\mathbf{S}_1 = (\{\mathrm{N}_1, \mathrm{N}_2\}, \{(\mathsf{o}_1, \mathrm{N}_1), (\mathsf{o}_2, \mathrm{N}_2)\})$ and $\mathbf{S}_2 = (\{\mathrm{N}_1, \mathrm{N}_2\}, \{(\mathsf{o}_1, \mathrm{N}_1), (\mathsf{o}_2, \mathrm{N}_1)\})$. Since $\mathbf{S}_1$ and $\mathbf{S}_2$ are 1-counting-equivalent, while $[\![\texttt{set-ids}]\!]_{\mathbf{S}_1} = \texttt{true}$ and $[\![\texttt{set-ids}]\!]_{\mathbf{S}_2} = \texttt{false}$, `set-ids` is not 1-counting-only. For a structure $\mathbf{S} = (\mathbf{N}, \gamma)$ with $|\mathbf{N}| = n$, $[\![\texttt{set-ids}]\!]_{\mathbf{S}} = \texttt{true}$ if and only if there exist $\mathsf{o}_1, \ldots, \mathsf{o}_n \in \mathrm{adom}(\mathbf{S})$ such that, for all i, $1 \leq i \leq n$, $[\![\mathrm{count}(\mathsf{o}_i)]\!]_{\mathbf{S}} = 1$ and, for all i, j, $1 \leq i < j \leq n$, $[\![\mathrm{count}(\mathsf{o}_i, \mathsf{o}_j)]\!]_{\mathbf{S}} = 0$. By Proposition 2.8, `set-ids` is 2-counting-only. □

Observe that `set-ids` can only evaluate to `true` on a structure if the size of its active domain is lowerbounded by the number of set names in the structure. This contradicts `set-ids` being expressible in SimpleCALC provided we can prove that whenever a SimpleCALC query evaluates to `true` on some structure, it also evaluates to `true` on some structure for which the size of the active domain is upperbounded by a function of the size of the query only. Thereto, we start with QuineCALC queries. If a QuineCALC query returns on some structure the tuple t, we can intuitively reduce the number of active-domain objects in that structure to the number of object variables in the query without compromising that t is returned, because all object variables are free. In order to substantiate this intuition, we introduce the notion of *active-domain preservation*:

Definition 5.6. *Let* $\mathbf{S} = (\mathbf{N}, \gamma)$ *be a structure and* I *an itemset. A bijection* $m : \mathcal{D} \to \mathcal{D}$ *is* active-domain preserving *for* $\mathbf{S}$ *and* I *if it is the identity on* $\mathrm{adom}(\mathbf{S}|_{\mathrm{I}})$*, and maps objects to* $\mathcal{D} - \mathrm{adom}(\mathbf{S}|_{\mathrm{I}})$ *only if they are in* $\mathcal{D} - \mathrm{adom}(\mathbf{S})$.

Notice that m is not necessarily the identity on all of I.

For QuineCALC queries with k (free) object variables, we can use active-domain preservation to state in a precise way that, for our purposes, we can restrict the active domain of structures to k objects:

Proposition 5.7. *Let* e *be a partial QuineCALC formula with* k *object variables. For every structure* $\mathbf{S} = (\mathbf{N}, \gamma)$*, mapping* $\nu_{\mathcal{D}}$ *from object variables in* e *to an itemset* $\mathrm{I} \subset \mathcal{D}$ *with* $|\mathrm{I}| \leq k$*, mapping* $\nu_{\mathbf{N}}$ *from free set name variables in* e *to* $\mathbf{N}$*, and active-domain preserving mapping* m *for* $\mathbf{S}$ *and* I*,* $(\mathbf{S}, \nu_{\mathcal{D}}, \nu_{\mathbf{N}}) \vDash e \iff (\mathbf{S}|_{\mathrm{I}}, m \circ \nu_{\mathcal{D}}, \nu_{\mathbf{N}}) \vDash e$.

To generalize Proposition 5.7 to SimpleCALC, we need to take into account object quantification:

Definition 5.8. *Let e be a SimpleCALC query. We denote the* object variable count *of e by* $\mathrm{vars}(e)$*. If e is a QuineCALC query with k (free) object variables, then* $\mathrm{vars}(e) = k$*; if* $e \equiv \neg e'$ *or* $e \equiv \exists x\ e'$*, then* $\mathrm{vars}(e) = \mathrm{vars}(e')$*; and if* $e \equiv e_1 \vee e_2$*, then* $\mathrm{vars}(e) = \mathrm{vars}(e_1) + \mathrm{vars}(e_2)$*.*

Proposition 5.9. *Let e be a SimpleCALC query with k free object variables, $\mathbf{S} = (\mathbf{N}, \gamma)$ a structure, and $\nu_{\mathcal{D}}$ a mapping from free object variables in e to an itemset $\mathrm{I} \subset \mathcal{D}$ with $|\mathrm{I}| \leq k$. There exists an itemset V with $\mathrm{I} \subseteq \mathrm{V}$ and $|\mathrm{V}| \leq \mathrm{vars}(e)$ such that, for every itemset W with $\mathrm{V} \subseteq \mathrm{W}$ and active-domain preserving mapping m for $\mathbf{S}$ and W, we have $(\mathbf{S}, \nu_{\mathcal{D}}, \emptyset) \vDash e$ if and only if $(\mathbf{S}|_{\mathrm{W}}, m \circ \nu_{\mathcal{D}}, \emptyset) \vDash e$.*

We can now prove that `set-ids` is not expressible in SimpleCALC:

Proposition 5.10. *The query* `set-ids` *is not expressible in SimpleCALC.*

Proof. Assume there exists a (Boolean) SimpleCALC query e such that, for every structure $\mathbf{S}$, $[\![e]\!]_{\mathbf{S}} = [\![\texttt{set-ids}]\!]_{\mathbf{S}}$. Let $n = \mathrm{vars}(e)+1$, $\{\mathsf{o}_0, \ldots, \mathsf{o}_n\}$ an itemset, and $\mathbf{N} = \{\mathrm{N}_0, \ldots, \mathrm{N}_n\} \subset \mathcal{N}$. Let $\mathbf{S}_{n+1} = (\mathbf{N}, \{(\mathsf{o}_i, \mathrm{N}_i) \mid 0 \leq i \leq n+1\})$, and $\mathbf{S}_n = (\mathbf{N}, \{(\mathsf{o}_i, \mathrm{N}_i) \mid 1 \leq i \leq n\})$. Hence, $\mathbf{S}_n = \mathbf{S}_{n+1}|_{\mathrm{W}}$ with $\mathrm{W} = \{\mathsf{o}_1, \ldots, \mathsf{o}_n\}$. By construction, $[\![e]\!]_{\mathbf{S}_{n+1}} \neq \emptyset$ and $[\![e]\!]_{\mathbf{S}_n} = \emptyset$. By Proposition 5.9, however, $[\![e]\!]_{\mathbf{S}_{n+1}} = \emptyset \iff [\![e]\!]_{\mathbf{S}_n} = \emptyset$, a contradiction. Hence, `set-ids` is not expressible in SimpleCALC. □

Corollary 5.11. *There is a Boolean 2-SyCALC query not expressible in SimpleCALC.*

6 Dichotomy for satisfiability-related decision problems

We study the decidability of satisfiability, validity, query containment, and query equivalence for the query languages we introduced. We first observe the following:

Lemma 6.1. *Let L be k-SyCALC or SimpleCALC-k, and p_1 and p_2 two decision problems chosen from satisfiability, validity, query containment, and query equivalence. Then p_1 is decidable for L if and only if p_2 is decidable for L.*

Because of Lemma 6.1, we only study the satisfiability problem in more detail.

6.1 Satisfiability of SimpleCALC is decidable

To prove that satisfiability is decidable for queries in SimpleCALC, we show that this language has the *finite model property*: a query is satisfiable if and only if it is satisfiable in a structure of which the size (in terms of the number of set names and active domain objects) is uniformly bounded in terms of the size of the query. Proposition 5.9 gives an upperbound on the required number of active domain objects. To also obtain an upperbound on the required number of set names, we consider that SyCALC is essentially a two-sorted variant of first-order logic. Intuitively, this puts severe restrictions to the ability of SyCALC and SimpleCALC to count. We formalize this intuition next.

Definition 6.2. *Let $k, d \geq 0$. Structures $\mathbf{S}_1 = (\mathbf{N}_1, \gamma_1)$ and $\mathbf{S}_2 = (\mathbf{N}_2, \gamma_2)$ are d-*partial k-counting-equivalent *if, for every pair of itemsets I and E with $|\mathrm{I} \cup \mathrm{E}| \leq k$, either*

(i) $[\![\mathrm{gcount}(\mathrm{I}; \mathrm{E})]\!]_{\mathbf{S}_1} = [\![\mathrm{gcount}(\mathrm{I}; \mathrm{E})]\!]_{\mathbf{S}_2} \leq d$; *or*
(ii) $d < [\![\mathrm{gcount}(\mathrm{I}; \mathrm{E})]\!]_{\mathbf{S}_1} < |\mathbf{N}_1| - d$ *and* $d < [\![\mathrm{gcount}(\mathrm{I}; \mathrm{E})]\!]_{\mathbf{S}_2} < |\mathbf{N}_2| - d$; *or*
(iii) $|\mathbf{N}_1| - [\![\mathrm{gcount}(\mathrm{I}; \mathrm{E})]\!]_{\mathbf{S}_1} = |\mathbf{N}_2| - [\![\mathrm{gcount}(\mathrm{I}; \mathrm{E})]\!]_{\mathbf{S}_2} \leq d$.

Even though partial counting-equivalence is a weaker condition than counting-equivalence, it is nevertheless sufficient to establish the indistinguishability of two structures by a SyCALC query if we know its set name quantifier depth:

Lemma 6.3. *Let e be a SyCALC query with set name quantifier depth d, and let $\mathbf{S}_1$ and $\mathbf{S}_2$ be d-partial k-counting-equivalent structures with $k = |\mathrm{adom}(\mathbf{S}_1)| = |\mathrm{adom}(\mathbf{S}_2)|$. Then $[\![e]\!]_{\mathbf{S}_1} = [\![e]\!]_{\mathbf{S}_2}$.*

Lemma 6.3 can be proved using an Ehrenfeucht-Fraïssé game in which the Spoiler can play up to d set names and an arbitrary number of objects. We now use this lemma to prove the following upperbound:

Proposition 6.4. *Let $\mathbf{S} = (\mathbf{N}, \gamma)$ be a structure with $|\mathrm{adom}(\mathbf{S})| = k$, and let $d \geq 0$. There exists a structure $\mathbf{S}' = (\mathbf{N}', \gamma')$ with $|\mathbf{N}'| \leq (d+1) \cdot 2^k$ such that $\mathbf{S}$ and $\mathbf{S}'$ are d-partial k-counting-equivalent structures.*

Proof. (Sketch.) Initially, $\mathbf{S}'$ is empty. Then, for every itemset I of $\mathbf{S}$, we add $\min(d+1, [\![\mathrm{gcount}(\mathrm{I}; \mathrm{adom}(\mathbf{S}) - \mathrm{I})]\!]_{\mathbf{S}})$ relation names to $\mathbf{N}'$ and associate each of them in γ' with precisely all elements of I. By construction, $|\mathbf{N}'| \leq (d+1) \cdot 2^k$. It is then verified that $\mathbf{S}$ and $\mathbf{S}'$ are d-partial k-counting-equivalent. □

Combining Propositions 5.9 and 6.4 proves that SimpleCALC has the finite model property and that the size of these finite models is uniformly upperbounded by an exponential function of the query size. Hence, the satisfiability problem is decidable. Using a reduction involving monadic first-order logic (over structures with only unary relations), for which satisfiability is NEXPTIME-complete [14,3], we can also prove a lowerbound on the complexity of the satisfiability problem:

Theorem 6.5. *Satisfiability is decidable for SimpleCALC queries, and is NEXPTIME-hard for SimpleCALC-k query, $k \geq 2$.*

Proof. (Sketch.) Let $S = (\mathcal{M}; X_1, \ldots, X_n)$ be a first-order structure over domain $\mathcal{M}$ with unary predicates $X_1, \ldots, X_n$ and φ a first-order logic formula over S without free variables. We encode the first-order structure S into bag-of-sets structure. To do so, we represent the unary predicates $X_1, \ldots, X_n$ by set names $\mathrm{N}_1, \ldots, \mathrm{N}_n$. In SimpleCALC, we cannot freely use set name quantification, however. We solve this by associating to each set name N_i a unique identifying object o_i, $1 \leq i \leq n$. The domain element of $\mathcal{M}$ are represented by objects distinct from $\mathsf{o}_1, \ldots, \mathsf{o}_n$, and translate predicate membership tests into count$(\cdot, \cdot)$ terms.

In summary, we encode S by a structure $\mathbf{S} = (\mathbf{N}, \gamma)$ with $\mathbf{N} = \{\mathrm{N}_1, \ldots, \mathrm{N}_n\}$ and $\gamma = \{(\mathsf{o}_1, \mathrm{N}_1), \ldots, (\mathsf{o}_n, \mathrm{N}_n)\} \cup \{(\mathsf{m}, \mathrm{N}_i) \mid m \in \mathcal{M} \wedge X_i(m)\}$, in which m is the object representing m. We now translate φ to the expression e given by

$$\mathrm{count}() = n \wedge \exists y_1 \ldots \exists y_n \Big(\tau(\varphi) \wedge \big(\bigwedge_{1 \leq i \leq n} \mathrm{count}(y_i) = 1\big) \wedge \big(\bigwedge_{1 \leq i < j \leq n} \mathrm{count}(y_i, y_j) = 0\big)\Big),$$

in which $\tau(\varphi)$ is the translation of φ obtained by replacing all subformula $\exists y\, \varphi'$ by $\exists y\, (\bigwedge_{1 \leq i \leq n}(y \neq y_i) \wedge \tau(\varphi'(y))$ and all terms of the form $X_i(b)$ by $\mathrm{count}(b, y_i) = 1$. Using Lemma 6.10, one can prove that the resulting Boolean formula e is in SimpleCALC-2, and that e is satisfiable if and only if the monadic first-order logic formula φ is satisfiable. □

6.2 Satisfiability of 1-SyCALC is decidable

By Propositions 5.2, the decidability of the satisfiability problem for 0-SyCALC follows from the decidability of the satisfiability problem for SimpleCALC-1. This does not extend to 1-SyCALC, unfortunately, but we can still prove that the satisfiability problem for 1-SyCALC is decidable. Again, we show that the finite model property holds. First, we put an upperbound on the number of set names.

Proposition 6.6. *Let $d \geq 0$, and let $\mathbf{S} = (\mathbf{N}, \gamma)$ be a structure. There exists a structure $\mathbf{S}' = (\mathbf{N}', \gamma')$ with $|\mathbf{N}'| \leq 2d + 1$ such that $\mathbf{S}$ and $\mathbf{S}'$ are d-partial 1-counting-equivalent structures.*

Proof. If $|\mathbf{N}| \leq 2d + 1$, we put $\mathbf{S}' = \mathbf{S}$, and Proposition 6.6 trivially holds. Otherwise, let $\mathbf{N}' = \{\mathrm{N}_1, \ldots, \mathrm{N}_{2d+1}\}$ and

$$\begin{aligned}\gamma' = {} & \{(\mathsf{o}, \mathrm{N}_i) \mid (\llbracket \mathrm{count}(\mathsf{o}) \rrbracket_{\mathbf{S}} \leq d) \wedge (1 \leq i \leq \llbracket \mathrm{count}(\mathsf{o}) \rrbracket_{\mathbf{S}})\} \cup \\ & \{(\mathsf{o}, \mathrm{N}_i) \mid (d < (\llbracket \mathrm{count}(\mathsf{o}) \rrbracket_{\mathbf{S}}) < |\mathbf{N}| - d) \wedge (1 \leq i \leq d + 1)\} \cup \\ & \{(\mathsf{o}, \mathrm{N}_i) \mid (|\mathbf{N}| - d \leq \llbracket \mathrm{count}(\mathsf{o}) \rrbracket_{\mathbf{S}}) \wedge (1 \leq i \leq 2d + 1 - (|\mathbf{N}| - \llbracket \mathrm{count}(\mathsf{o}) \rrbracket_{\mathbf{S}})\}.\end{aligned}$$

Using that, for $o \in \mathcal{D}$ and $\mathbf{S}'' = (\mathbf{N}'', \gamma'')$ any structure, $\llbracket \mathrm{gcount}(\mathsf{o}; \emptyset) \rrbracket_{\mathbf{S}''} = \llbracket \mathrm{count}(\mathsf{o}) \rrbracket_{\mathbf{S}''}$ and $\llbracket \mathrm{gcount}(\emptyset; \mathsf{o}) \rrbracket_{\mathbf{S}''} = |\mathbf{N}''| - \llbracket \mathrm{count}(\mathsf{o}) \rrbracket_{\mathbf{S}''}$, we can verify that $\mathbf{S}$ and $\mathbf{S}'$ are d-partial 1-counting-equivalent structures. □

Next, we put an upper bound on the number of objects.

Proposition 6.7. *Let e be a 1-SyCALC query with set name quantifier depth d and object quantifier depth r, and let $\mathbf{S} = (\mathbf{N}, \gamma)$ be a structure. Then, $\llbracket e \rrbracket_{\mathbf{S}} \neq \emptyset$ if and only if there exists a structure $\mathbf{S}' = (\mathbf{N}', \gamma')$ with $|\mathbf{N}'| \leq 2d+1$, $|\mathrm{adom}(\mathbf{S}')| \leq r(2d + 1)$, and $\llbracket e \rrbracket_{\mathbf{S}'} \neq \emptyset$.*

Proof. (Sketch.) By Proposition 6.6, we may assume without loss of generality that $|\mathbf{N}| \leq 2d+1$. Let $\mathbf{N}' = \{\mathrm{N}_1, \ldots, \mathrm{N}_{|\mathbf{N}|}\}$ and $\mathrm{I}_i = \{\mathsf{o} \mid [\![\mathrm{count}(\mathsf{o})]\!]_{\mathbf{S}} = i\}$, $1 \leq i \leq |\mathbf{N}|$. Since $\mathbf{S}$ and $\mathbf{S}'' = (\mathbf{N}', \gamma'')$ where $\gamma'' = \{(\mathsf{o}, \mathrm{N}_j) \mid (1 \leq j \leq i \leq |\mathbf{N}|) \wedge (\mathsf{o} \in \mathrm{I}_i)\}$ are 1-counting-equivalent, $[\![e]\!]_{\mathbf{S}''} = [\![e]\!]_{\mathbf{S}}$. Choose $\mathrm{P}_i \subseteq \mathrm{I}_i$ such that $|\mathrm{P}_i| = \min(|\mathrm{I}_i|, r)$, $1 \leq i \leq |\mathbf{N}|$, and let $\mathbf{S}' = (\mathbf{N}', \gamma')$ where $\gamma' = \{(\mathsf{o}, \mathrm{N}_j) \mid (1 \leq j \leq i \leq |\mathbf{N}|) \wedge (\mathsf{o} \in \mathrm{P}_i)\}$. We can show that e cannot distinguish between $\mathbf{S}'$ and $\mathbf{S}$ using an Ehrenfeucht-Fraïssé game in which the Spoiler can play up to r objects and an arbitrary number of set names. □

Propositions 6.6 and 6.7 combined prove that 1-SyCALC has the finite model property and that the size of these finite models is uniformly upperbounded by a polynomial function of the query size. Hence,

Theorem 6.8. *The satisfiability problem is decidable for* 1-*SyCALC queries.*

6.3 Satisfiability of 2-SyCALC is undecidable

To prove undecidability of satisfiability for 2-SyCALC, we reduce satisfiability of standard first-order logic queries on undirected unlabeled graphs without self-loops, a well-known undecidable problem,[10] to satisfiability of the strict fragment of 2-SyCALC that does not allow object comparisons (of the form $x = y$).

An *undirected unlabeled graph without self-loops*, or *graph*, for short, is a pair $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ in which $\mathbf{V}$ is a set of nodes and $\mathbf{E} \subseteq \mathbf{V} \times \mathbf{V}$ is an antireflexive and symmetric edge relation. On such graphs we consider standard first-order logic formulae of the form $e := x_1 = x_2 \mid \mathbf{E}(x_1, x_2) \mid e \vee e \mid \neg e \mid \exists x\ e$, in which x_1, x_2, and x are node variables. We write $[\![e]\!]_{\mathbf{G}}$ to denote the evaluation of e on $\mathbf{G}$.

We define the encoding of $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ as the structure $\mathrm{enc}(\mathbf{G}) = (\mathbf{N}, \gamma)$ where $\mathbf{N} = \mathbf{V}$ and $\gamma = \{(\{x_1, x_2\}, x_1), (\{x_1, x_2\}, x_2) \mid (x_1, x_2) \in \mathbf{E}\} \cup \{(\{x\}, x) \mid x \in \mathbf{V}\}$. The active domain consists of node-pair sets, representing the edges of $\mathbf{G}$, and singleton node sets, serving as distinctive identifying objects. Each node pair set has a support of 2, identifying the end-points of the edge represented. The structure $\mathrm{enc}(\mathbf{G})$ always satisfies the following Boolean SyCALC query:

$$\texttt{enc-graph} = \texttt{set-ids} \wedge \forall x \exists X_1 \exists X_2\ (((X_1 \neq X_2) \wedge \Gamma(x, X_1) \wedge \Gamma(x, X_2)) \Rightarrow \forall Y\ ((X_1 \neq Y) \wedge (X_2 \neq Y) \Rightarrow \neg\Gamma(x, Y))).$$

If ν converts node variables in a first-order logic formula on graphs φ, then the corresponding translation $\tau(\varphi)_\nu$ into a SyCALC query is defined as follows:

$$\begin{aligned} \tau(x_1 = x_2)_\nu &\equiv \nu(x_1) = \nu(x_2); \\ \tau(\mathbf{E}(x_1, x_2))_\nu &\equiv (\nu(x_1) \neq \nu(x_2)) \wedge \exists x\ (\Gamma(x, \nu(x_1)) \wedge \Gamma(x, \nu(x_2))); \\ \tau(e_1 \vee e_2)_\nu &\equiv \tau(e_1)_\nu \vee \tau(e_2)_\nu; \end{aligned}$$

[10] We have no direct reference, but if we use a straightforward encoding from binary relations to undirected unlabeled graphs without self-loops, we can rely on Trakhtenbrot's Theorem [15, Theorem 9.2].

$$\tau(\neg e)_\nu \equiv \neg\tau(e)_\nu;$$
$$\tau(\exists x\ e)_\nu \equiv \exists X\ \tau(e)_{\nu[x\mapsto X]},$$

with X a fresh set name variable. We define the encoding of a *Boolean* first-order logic formula on graphs φ in SyCALC as $\text{enc}(\varphi) = \texttt{enc-graph} \wedge \tau(\varphi)_\emptyset$. Obviously,

Lemma 6.9. *Let* $\mathbf{G}$ *be a graph and let* φ *be a Boolean first-order logic formula on graphs. Then,* $[\![\varphi]\!]_{\mathbf{G}} = [\![\text{enc}(\varphi)]\!]_{\text{enc}(\mathbf{G})}$.

Next, we prove that, for any first-order Boolean logic formula φ on graphs, $\text{enc}(\varphi)$ is a Boolean 2-SyCALC query. We do so by proving that 2-counting-equivalent structures satisfying the Boolean 2-SyCALC query `set-ids` must be isomorphic.

Lemma 6.10. *If* $\mathbf{S}_1$ *and* $\mathbf{S}_2$ *are structures that are* 2*-counting-equivalent, and* $[\![\texttt{set-ids}]\!]_{\mathbf{S}_1} = [\![\texttt{set-ids}]\!]_{\mathbf{S}_2} = \texttt{true}$*, then* $\mathbf{S}_1$ *and* $\mathbf{S}_2$ *are isomorphic.*

Corollary 6.11. *If* φ *is a Boolean first-order logic formula on graphs, then* $\text{enc}(\varphi)$ *is a* 2*-SyCALC query.*

Now, let $\mathbf{S}$ be a structure for which $[\![\text{enc}(\varphi)]\!]_{\mathbf{S}} \neq \emptyset$, with φ a Boolean first-order logic formula. For the last step in our reduction, we must find a graph $\mathbf{G}_\mathbf{S}$ such that $[\![\varphi]\!]_{\mathbf{G}_\mathbf{S}} \neq \emptyset$. Ideally, we would like that, up to isomorphism, $\text{enc}(\mathbf{G}_\mathbf{S}) = \mathbf{S}$, but that can unfortunately not be guaranteed. Nevertheless, we can construct a graph $\mathbf{G}_\mathbf{S}$ for which $[\![\varphi]\!]_{\mathbf{G}_\mathbf{S}} \neq \emptyset$:

Lemma 6.12. *Let* φ *be a Boolean first-order logic formula on graphs. If there exists a structure* $\mathbf{S}$ *satisfying* $\text{enc}(\varphi)$*, then we can construct from* $\mathbf{S}$ *a graph satisfying* φ*.*

Using Lemmas 6.9 and 6.12, we conclude the following:

Theorem 6.13. *The satisfiability problem is undecidable for* 2*-SyCALC queries.*

7 Conclusion and discussion

In this paper, we studied so-called counting-only queries on bag-of-sets data, which can be answered by only counting the occurrence of itemsets of objects. In particular, we identified and studied the syntactic counting-only fragments QuineCALC and SimpleCALC of first-order logic. These query languages can express many practically relevant queries other than the usual classes of "well-behaved" first-order queries—such as the conjunctive queries, the monadic first-order logic, and the two-variable fragments of first-order logic—while, at the same time, still being simple enough for satisfiability, validity, query containment, and query equivalence to be decidable. We have summarized our findings in Figure 3.

We have identified several directions for future research:

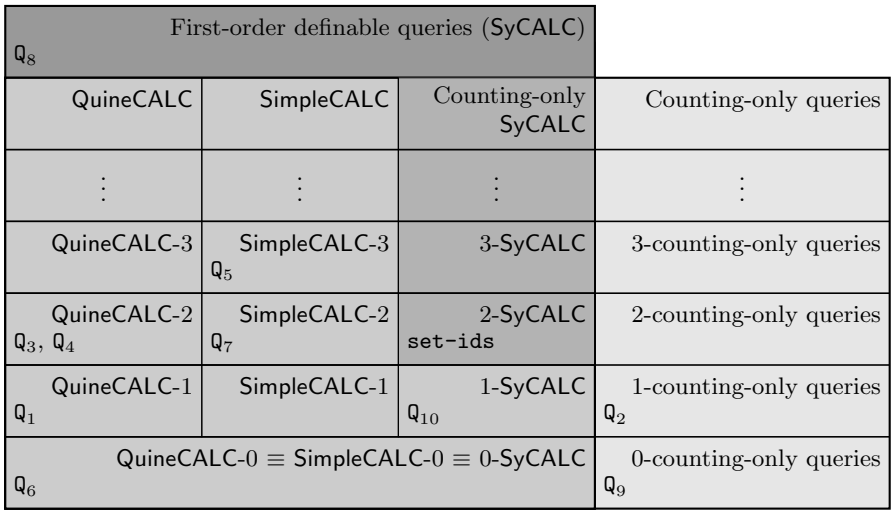


Fig. 3. Main relationships between the query languages considered. The counting-only languages are highlighted in light-grey, and the first-order definable languages in dark-grey. A language to the left and/or below another language, is less expressive than the latter. Separate boxes also indicate strict separation in expressive power. The example queries $\mathtt{Q}_1$–$\mathtt{Q}_6$ (Introduction), $\mathtt{Q}_7$ (Example 2.6), $\mathtt{Q}_8$ (proof of Proposition 3.3), $\mathtt{Q}_9$ (proof of Proposition 5.3), and $\mathtt{Q}_{10}$ (proof of Proposition 5.4) are added to the smallest language in which they can be expressed. The medium-dark grey area indicates the first-order definable counting-only queries for which satisfiability is not decidable.

1. In this paper, we have studied the formal aspect of counting-only first-order queries, but we have not yet studied practical issues such as query evaluation. Since the queries we study are all first-order queries, we can, off course, borrow standard techniques from first-order logic for their evaluation. One may wonder, however, if some of the more restricted classes considered in this paper allow for more efficient query evaluation, for example by using specialized counting-only index structures.

As an example, consider queries using *generalized count-term predicates*, which are all expressible in SimpleCALC. Queries based on generalized count-term predicates provide a direct connection to an underlying frequent itemset problems, which can be exploited to further optimize query equivalence. A good example of such a technique is the FP-tree, used by the FP-Growth Algorithm, which can be used as an index to quickly find candidate sets of up-to-k-objects that have a minimum count [12,6], and prune away all other sets of up-to-k-objects without any counting. Due to these implementation optimization opportunities and the prevalence of counting-only queries, we believe that the evaluation of these simple counting-only queries and their relationship to frequent itemset mining deserves a deeper understanding.

2. In the Introduction, we have already mentioned that the bag-of-sets data model and the notion of counting-only query can easily be generalized, e.g., to

a model with relations between more than two disjoint domains. Therefore, it is only natural to wonder if the concepts we developed generalize to a richer data model without giving up on the well-behaved nature of SimpleCALC.

3. From a more theoretical perspective, there are several open problems for further investigation. For example, the precise complexity of the decision problems for SimpleCALC-k, $k \geq 0$, remains open. Crucial in pinpointing an exact upperbound is finding the exact upperbound on the complexity of model checking. We also want to study the decidability of whether a given $(k+1)$-counting-only query is also k-counting-only.

4. Counting is only one type of measure that can be used to define practical queries on bag-of-sets data, and we have seen that taking counting into account leads to naturally definable and well-behaved query languages. Many other practical types of measure exist [18], hence it is only natural to ask if these measures can be captured in an encompassing framework that leads, for each measure, to naturally definable and well-behaved query languages.

5. As we have shown in this paper, not all counting-only queries are first-order definable. To express some of these queries, one might consider augmenting first-order logic with non-first-order definable counting-based quantifiers [13]. We believe that it is worthwhile to study whether one can construct such query languages while, at the same time, retain the well-behaved nature of SimpleCALC.

References

1. Abiteboul, S., Hull, R., Vianu, V. (Eds.): Foundations of Databases: The Logical Level. Addison-Wesley (1995)
2. Anderson, I.: Combinatorics of Finite Sets. Dover Publications (2011)
3. Bachmair, L., Ganzinger, H., Waldmann, U.: Set constraints are the monadic class. In: Proc. 8th Ann. IEEE Symp. on Logic in Computer Science. pp. 75–83 (1993)
4. Badia, A., Van Gucht, D., Gyssens, M.: Querying with Generalized Quantifiers, pp. 235–258. Springer US, Boston, MA (1995)
5. Bayer, A.E., Smart, J.C., McLaughlin, G.W.: Mapping intellectual structure of a scientific subfield through author cocitations. J. Am. Soc. Inform. Sci. Tech. 41(6), 444452 (1990)
6. Calders, T., Goethals, B.: Non-derivable itemset mining. Data Min. Knowl. Discov. 14(1), 171–206 (2007)
7. Fletcher, G.H.L., Van Den Bussche, J., Van Gucht, D., Vansummeren, S.: Towards a theory of search queries. ACM Trans. Database Syst. 35(4), 28:1–28:33 (2010)
8. Goethals, B.: Survey on frequent pattern mining. Tech. Rep., Univ. of Helsinki (2003)
9. Grädel, E., Otto, M.: On logics with two variables. Theor. Comput. Sci. 224(1–2), 73–113 (1999)
10. Grohe, M.: Finite variable logics in descriptive complexity theory. Bulletin of Symbolic Logic 4, 345–398 (1998)
11. Gyssens, M., Paredaens, J., Van Gucht, D., Wijsen, J., Wu, Y.: An approach towards the study of symmetric queries. Proc. VLDB Endow. 7(1), 25–36 (2013)

12. Han, J., Pei, J., Yin, Y., Mao, R.: Mining frequent patterns without candidate generation: A frequent-pattern tree approach. Data Min. and Knowl. Discov. 8(1), 53–87 (2004)
13. Kuske, D., Schweikardt, N.: First-order logic with counting. In: 32nd Ann. ACM/IEEE Symp. on Logic in Computer Science. pp. 1–12 (2017)
14. Lewis, H.R.: Complexity results for classes of quantificational formulas. J. Comput. Syst. Sci. 21(3), 317–353 (1980)
15. Libkin, L.: Elements of Finite Model Theory. Springer Berlin Heidelberg (2004)
16. Quine, W.V.: Selected Logic Papers. Harvard University Press (1995)
17. Sayrafi, B., Van Gucht, D.: Differential constraints. In: Proc. 24th Symp. on Principles of Database Systems. pp. 348–357. ACM (2005)
18. Sayrafi, B., Van Gucht, D., Gyssens, M.: Measures in databases and data mining. Tech. Rep. TR602, Indiana Univ. (2004), https://www.cs.indiana.edu/cgi-bin/techreports/TRNNN.cgi?trnum=TR602
19. Väänänen, J.: Generalized quantifiers, an introduction. In: Generalized Quantifiers and Computation: 9th European Summer School in Logic, Language, and Information. pp. 1–17. Springer Berlin Heidelberg (1999)